# Music Emotion Classification

Analyzing Sound and Sentiment

Data Mining & Machine Learning Project

# Four Key Features

## Four Key Features

Energy-Based Features

Rhythm and Tempo

Mel-Frequency Cepstral Coefficients (MFCCs)

Harmonic Features

## Acoustic Features in Real-World

Music Recommendation Systems

Speech Recognition and Emotion Detection

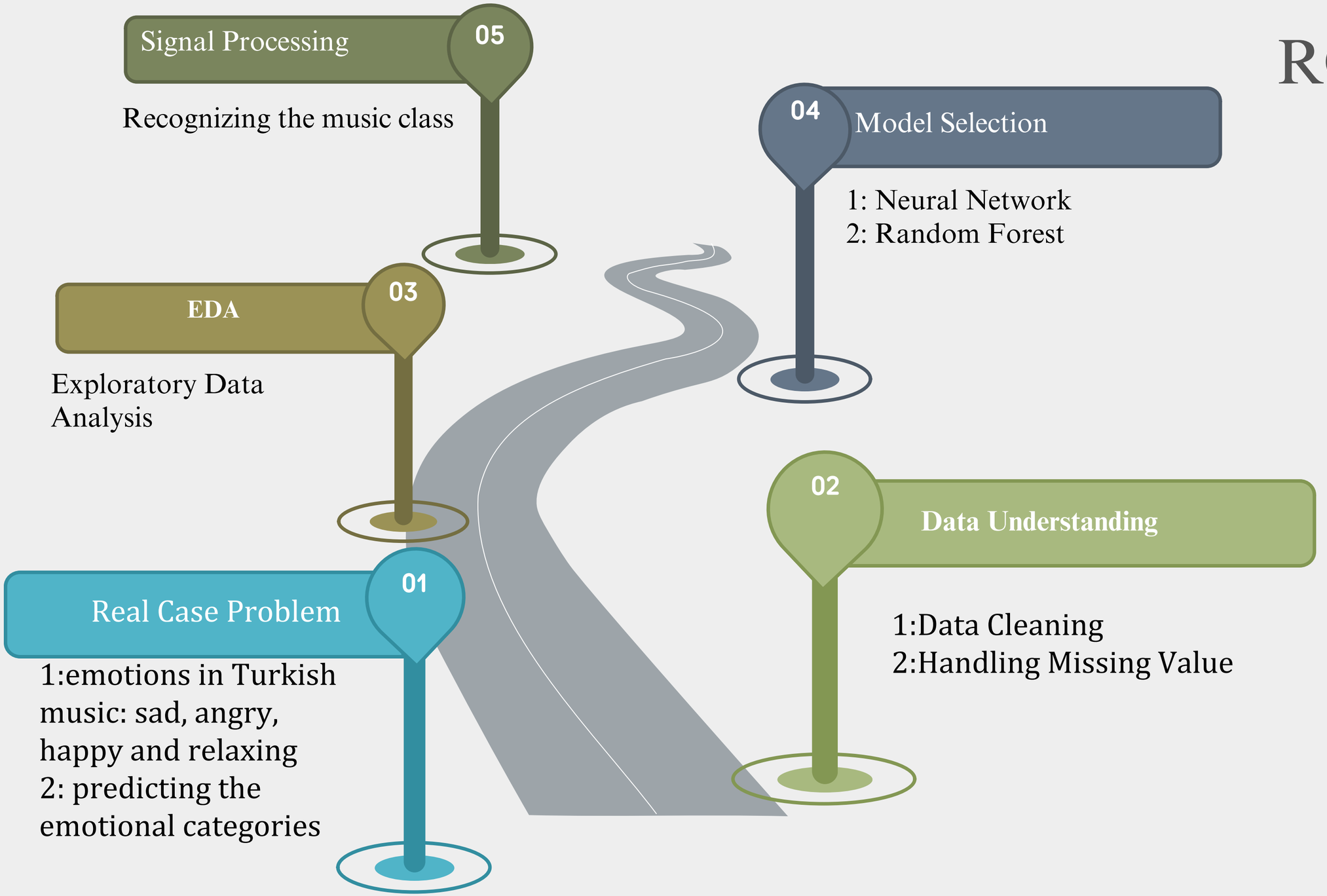
Environmental Sound Classification

# What is our goal?

Our goal is to build a model that accurately predicts the emotional category of Turkish songs.

- Sad
- Angry
- Happy
- Relaxing

# ROADMAP


Signal Processing
05
Recognizing the music class
04
Model Selection
1: Neural Network
2: Random Forest
EDA
03
Exploratory Data Analysis
02
Data Understanding
1:Data Cleaning
2:Handling Missing Value
Real Case Problem
01
1:emotions in Turkish music: sad, angry, happy and relaxing
2: predicting the emotional categories

# DATASET

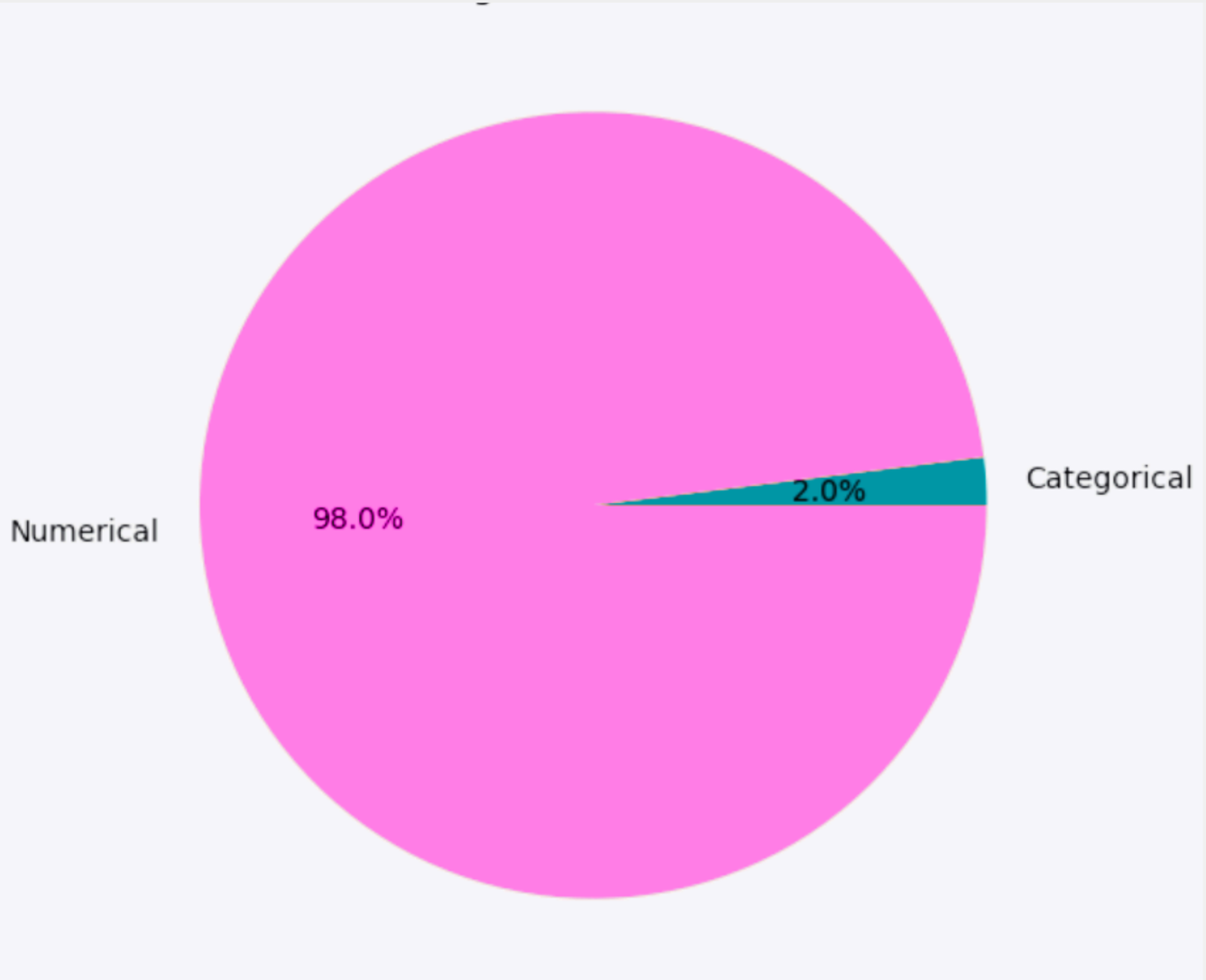
**Source**: The Article «*Music Emotion Recognition by Using Chroma Spectrogram and Deep Visual Features*»

**Website:** «*https://www.atlantis-press.com/journals/ijcis/125927469*»

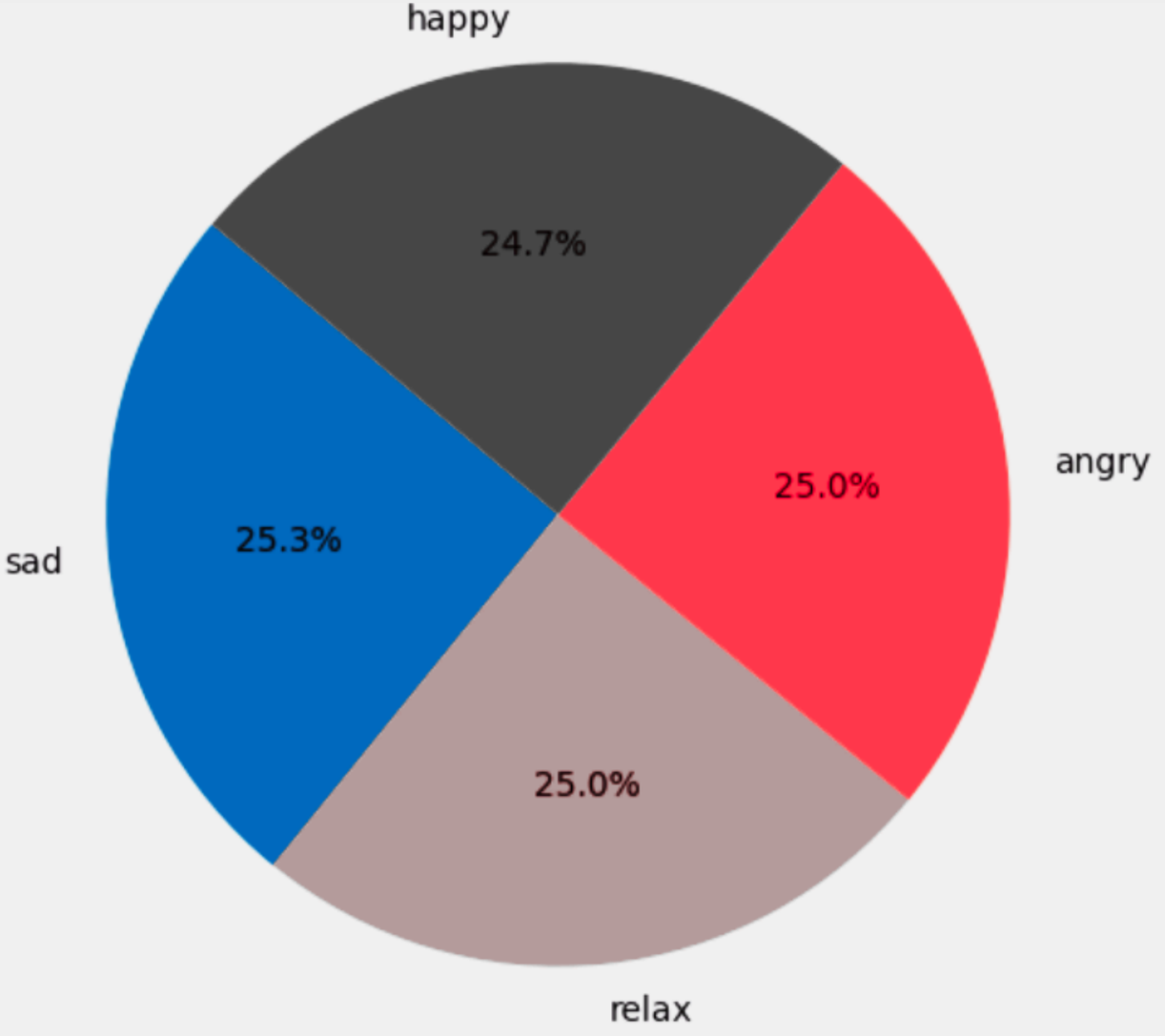
FEATURES

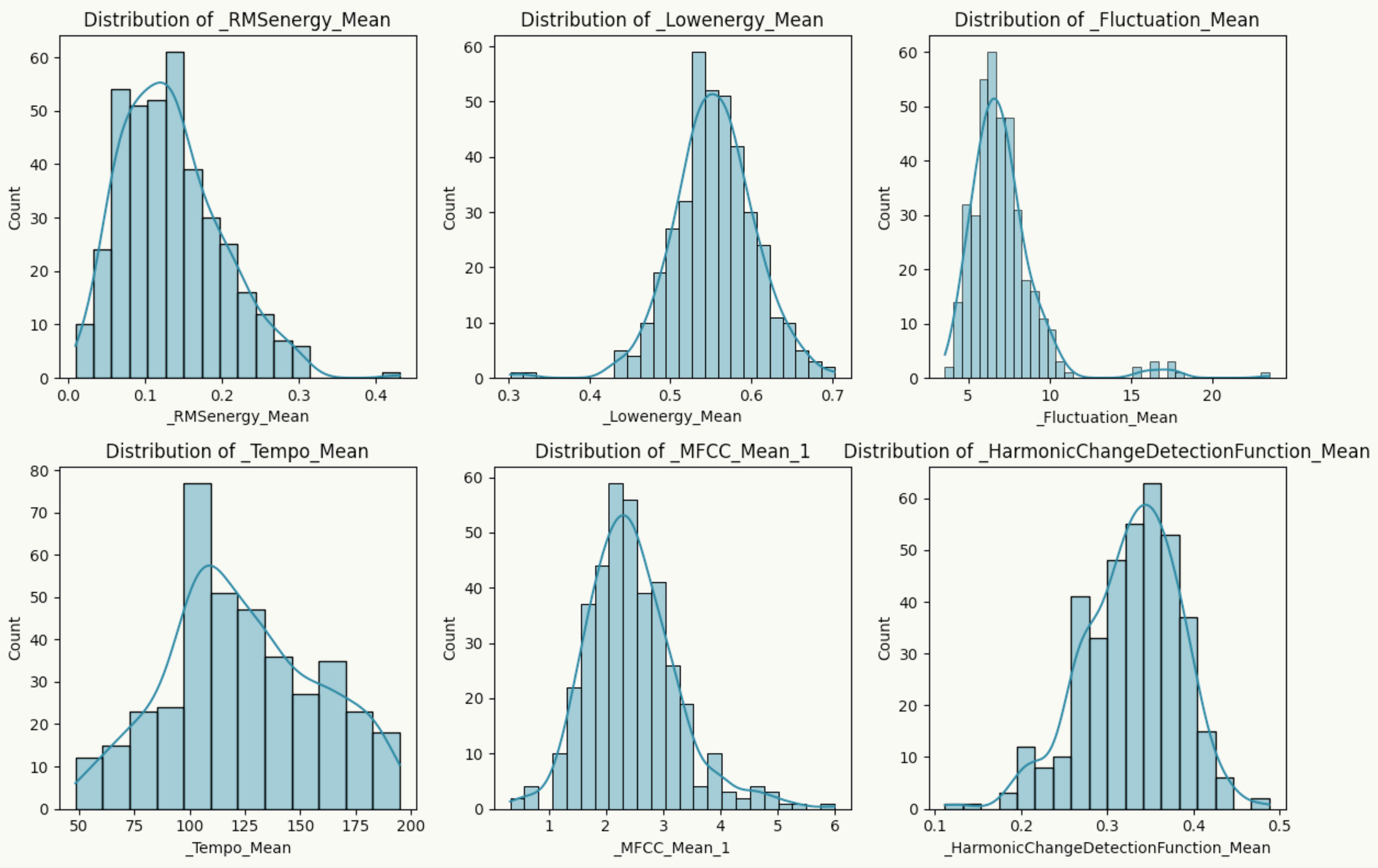
| *Categorical (1)* | *Numerical (50)* |
|---|---|
| Emotions | **RMSenergy Mean, Lowenergy Mean, Fluctuation Mean, Tempo Mean, MFCC Mean...** |

# The Percentage of Categorical and Numerical Data

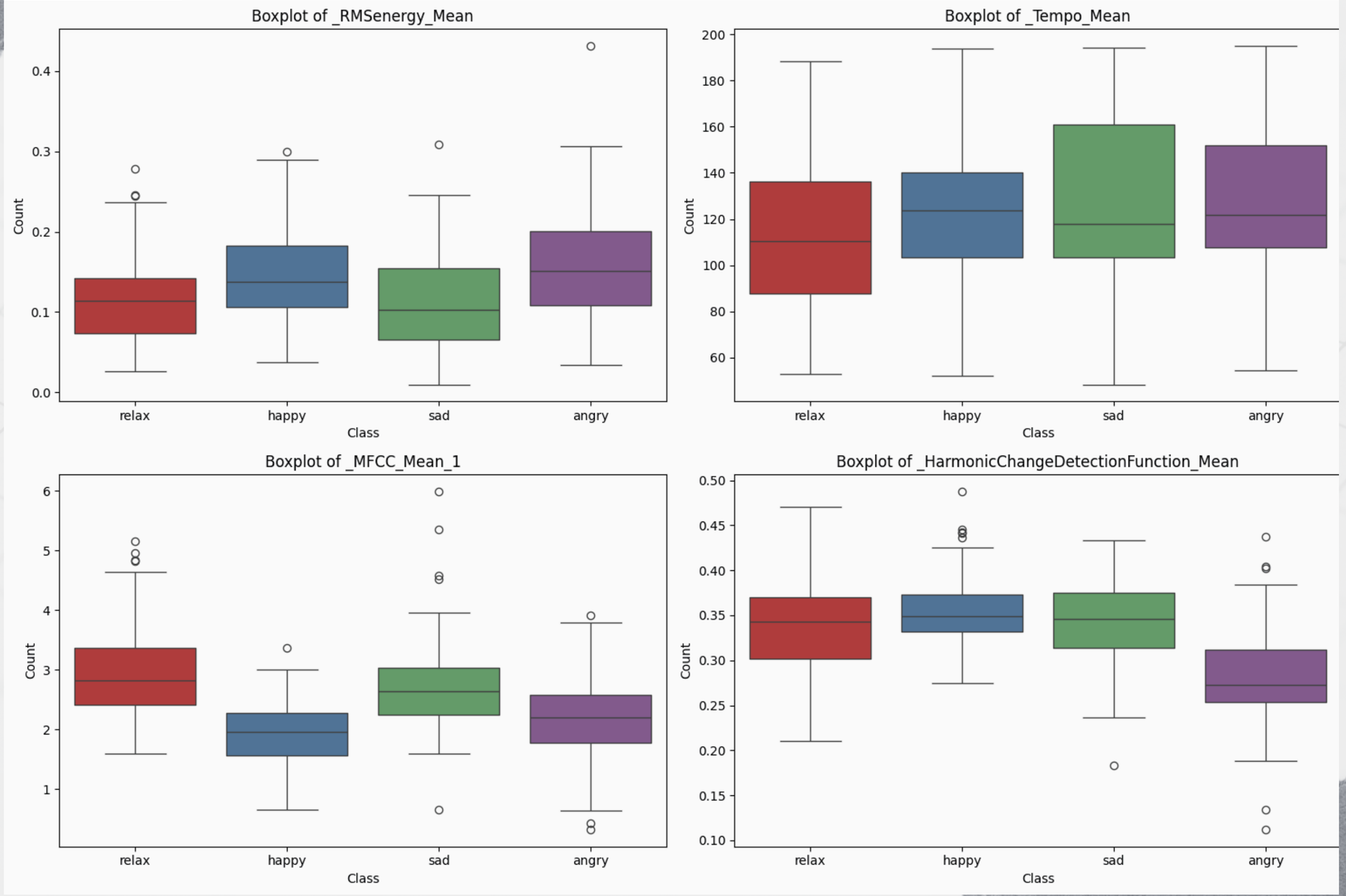



# Class Percentages

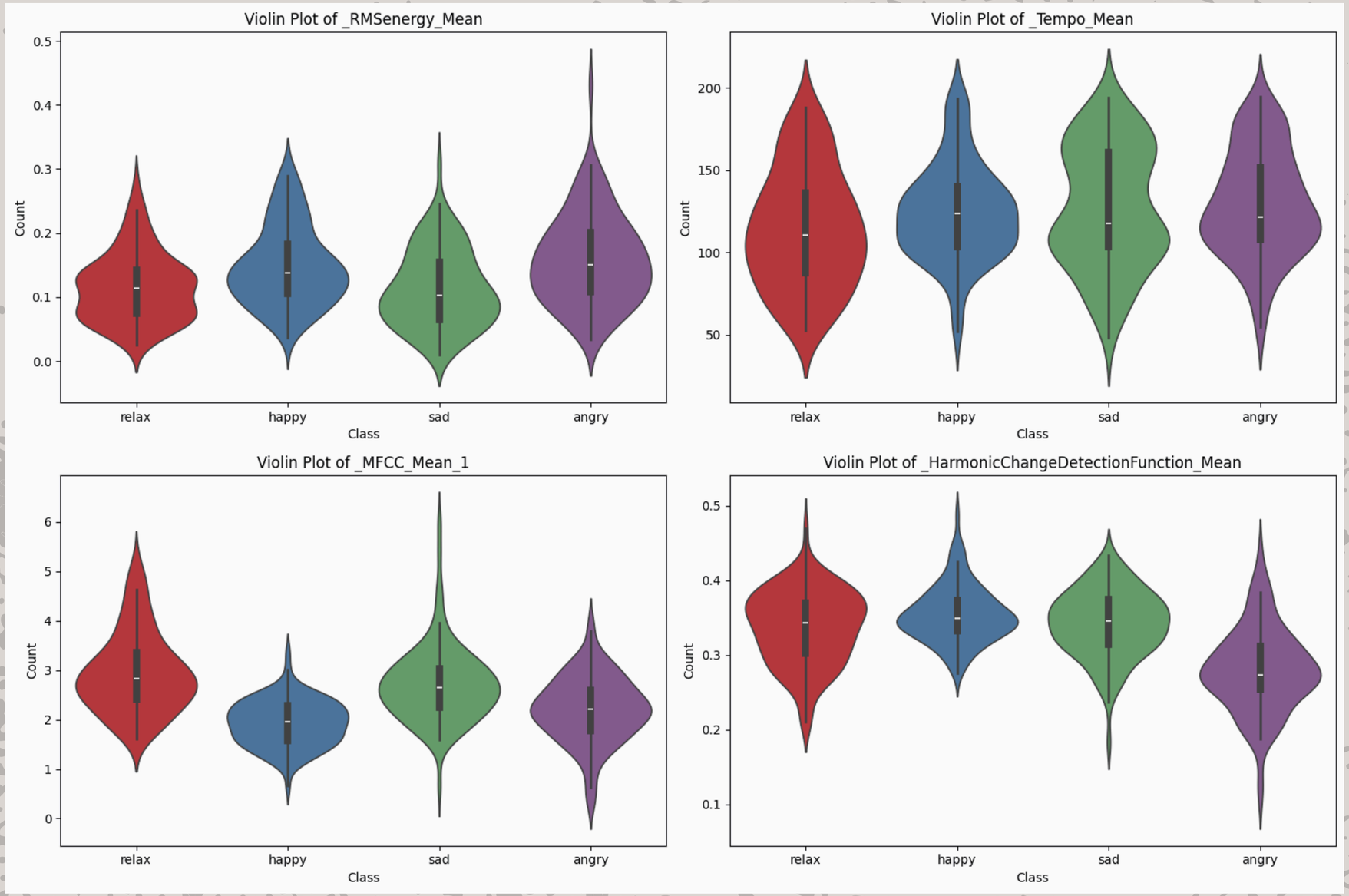



Distribution of _RMSenergy_Mean
Count
_RMSenergy_Mean
Distribution of _Lowenergy_Mean
Count
_Lowenergy_Mean
Distribution of _Fluctuation_Mean
Count
_Fluctuation_Mean
Distribution of _Tempo_Mean
Count
_Tempo_Mean
Distribution of _MFCC_Mean_1
Count
_MFCC_Mean_1
Distribution of _HarmonicChangeDetectionFunction_Mean
Count
_HarmonicChangeDetectionFunction_Mean

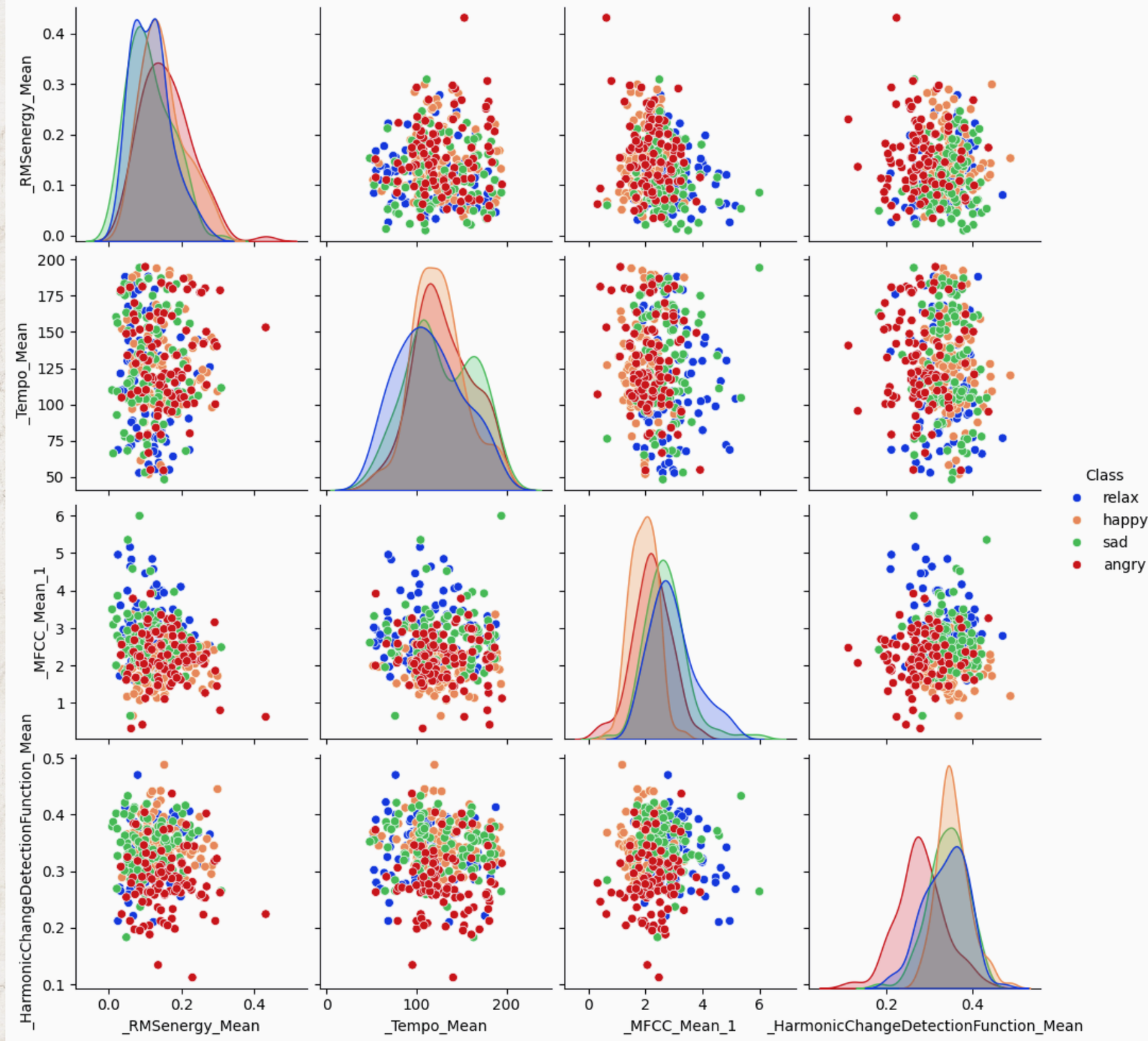
# Box Plot


Boxplot of _RMSenergy_Mean
Count
Class
relax
happy
sad
angry
0.0
0.1
0.2
0.3
0.4
Boxplot of _Tempo_Mean
Count
Class
relax
happy
sad
angry
60
80
100
120
140
160
180
200
Boxplot of _MFCC_Mean_1
Count
Class
relax
happy
sad
angry
1
2
3
4
5
6
Boxplot of _HarmonicChangeDetectionFunction_Mean
Count
Class
relax
happy
sad
angry
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.45
0.50

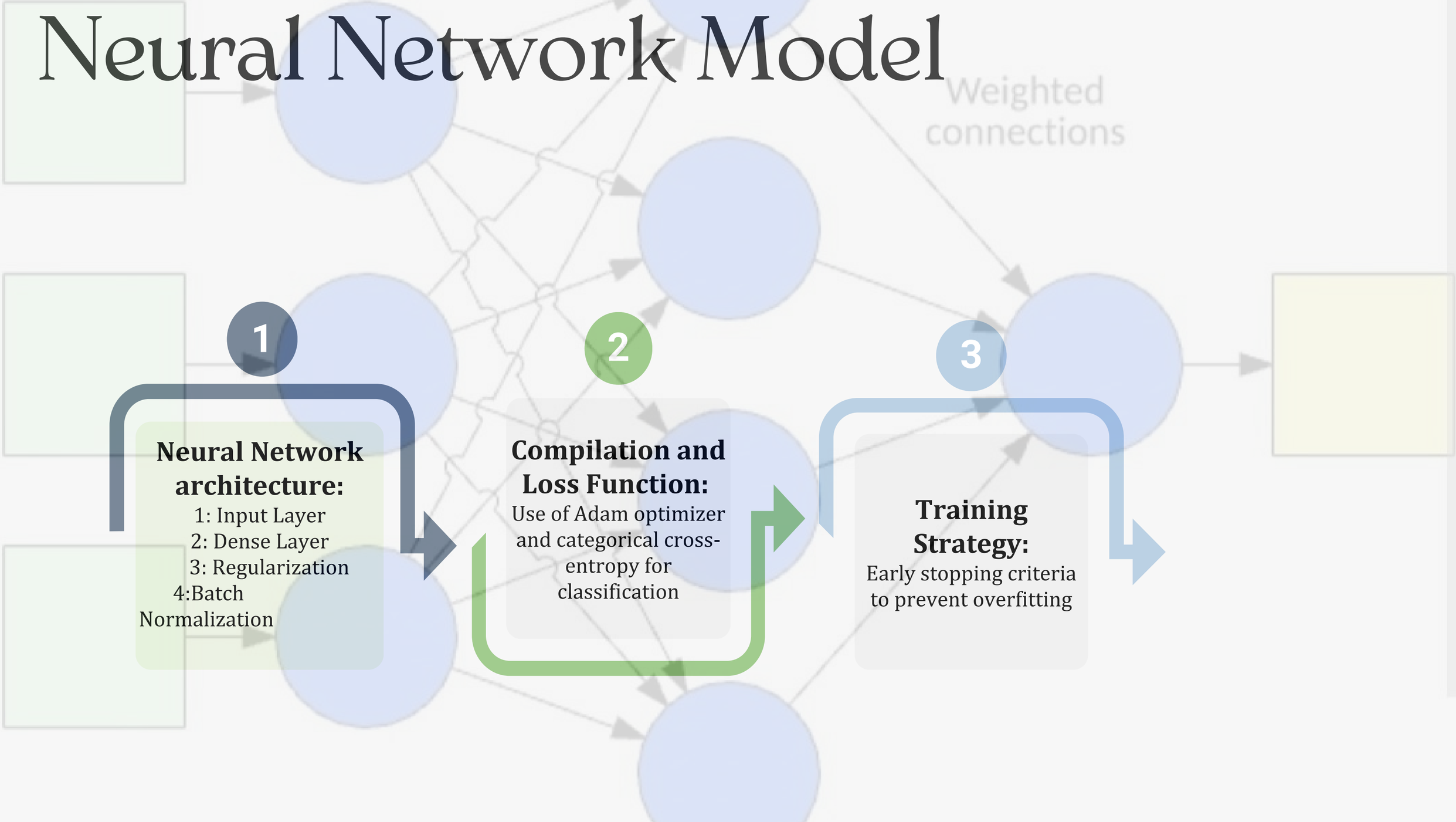
# Violin Plot


Violin Plot of _RMSenergy_Mean
Count
Class
relax
happy
sad
angry
Violin Plot of _Tempo_Mean
Violin Plot of _MFCC_Mean_1
Violin Plot of _HarmonicChangeDetectionFunction_Mean

# Pair Plot


_RMSenergy_Mean
_Tempo_Mean
_MFCC_Mean_1
_HarmonicChangeDetectionFunction_Mean
Class
relax
happy
sad
angry

# Neural Network Model




1

**Neural Network architecture:**

1: Input Layer
2: Dense Layer
3: Regularization
4:Batch Normalization

2

**Compilation and Loss Function:**

Use of Adam optimizer and categorical cross-entropy for classification

3

**Training Strategy:**

Early stopping criteria to prevent overfitting

# Data Splitting & Model Building

| Training: 80% | Testing: 20% |
|---|---|

```
# Build the Neural Network Model
model = Sequential()

# Layer 1: Add seed to kernel_initializer for reproducibility
model.add(Dense(256, input_dim=X_train.shape[1], activation='relu',
                kernel_regularizer=l2(0.001),
                kernel_initializer=tf.keras.initializers.GlorotUniform(seed=
model.add(BatchNormalization())
model.add(Dropout(0.3, seed=42))  # Set seed for Dropout

# Layer 2
model.add(Dense(128, kernel_regularizer=l2(0.001),
                kernel_initializer=tf.keras.initializers.GlorotUniform(seed=
model.add(LeakyReLU(alpha=0.1))
model.add(BatchNormalization())
model.add(Dropout(0.3, seed=42))  # Set seed for Dropout
```

```
# Layer 3
model.add(Dense(64, kernel_regularizer=l2(0.001),
                kernel_initializer=tf.keras.initializers.GlorotUniform(seed=42)))
model.add(LeakyReLU(alpha=0.1))
model.add(BatchNormalization())
model.add(Dropout(0.3, seed=42))  # Set seed for Dropout

# Output Layer
model.add(Dense(y_train_nn.shape[1], activation='softmax',
                kernel_initializer=tf.keras.initializers.GlorotUniform(seed=42)))
```

# Model Compiling & Training

```
[23] # Compile the model with the Adam optimizer
     optimizer = Adam(learning_rate=0.001)
     model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
```

```
[24] # Set early stopping (no randomness here)
     early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
```

```
[25] # Train the model
     history = model.fit(X_train, y_train_nn, epochs=100, batch_size=32, validation_data=(X_test, y_test_nn),
                         callbacks=[early_stopping], shuffle=False)  # Disable shuffling
```

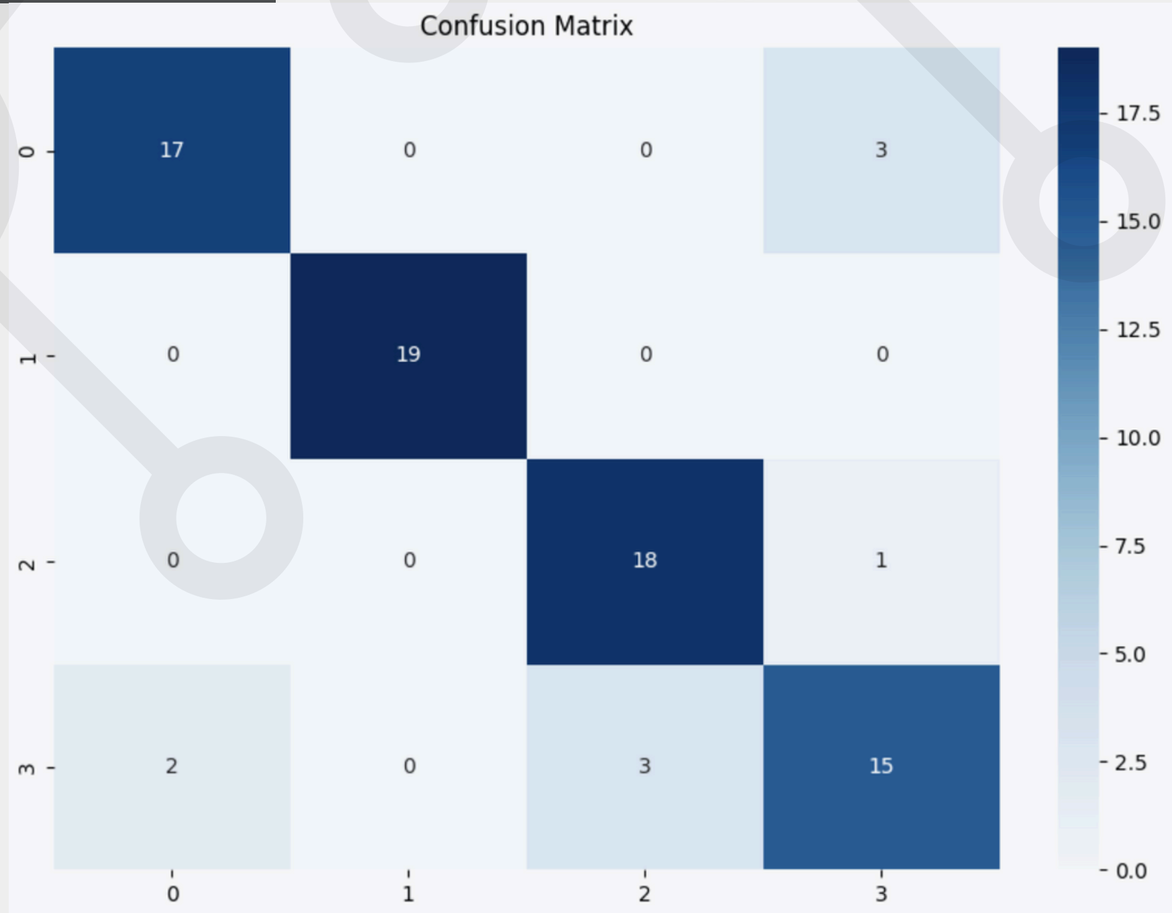
```
Epoch 41/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9886 - loss: 0.3684 - val_accuracy: 0.8462 - val_loss: 0.7175
Epoch 42/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9887 - loss: 0.3676 - val_accuracy: 0.8718 - val_loss: 0.7152
Epoch 43/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9983 - loss: 0.3483 - val_accuracy: 0.8718 - val_loss: 0.7172
Epoch 44/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9966 - loss: 0.3439 - val_accuracy: 0.8462 - val_loss: 0.7323
```

# Model Evaluation on the Test Data & Classification Report

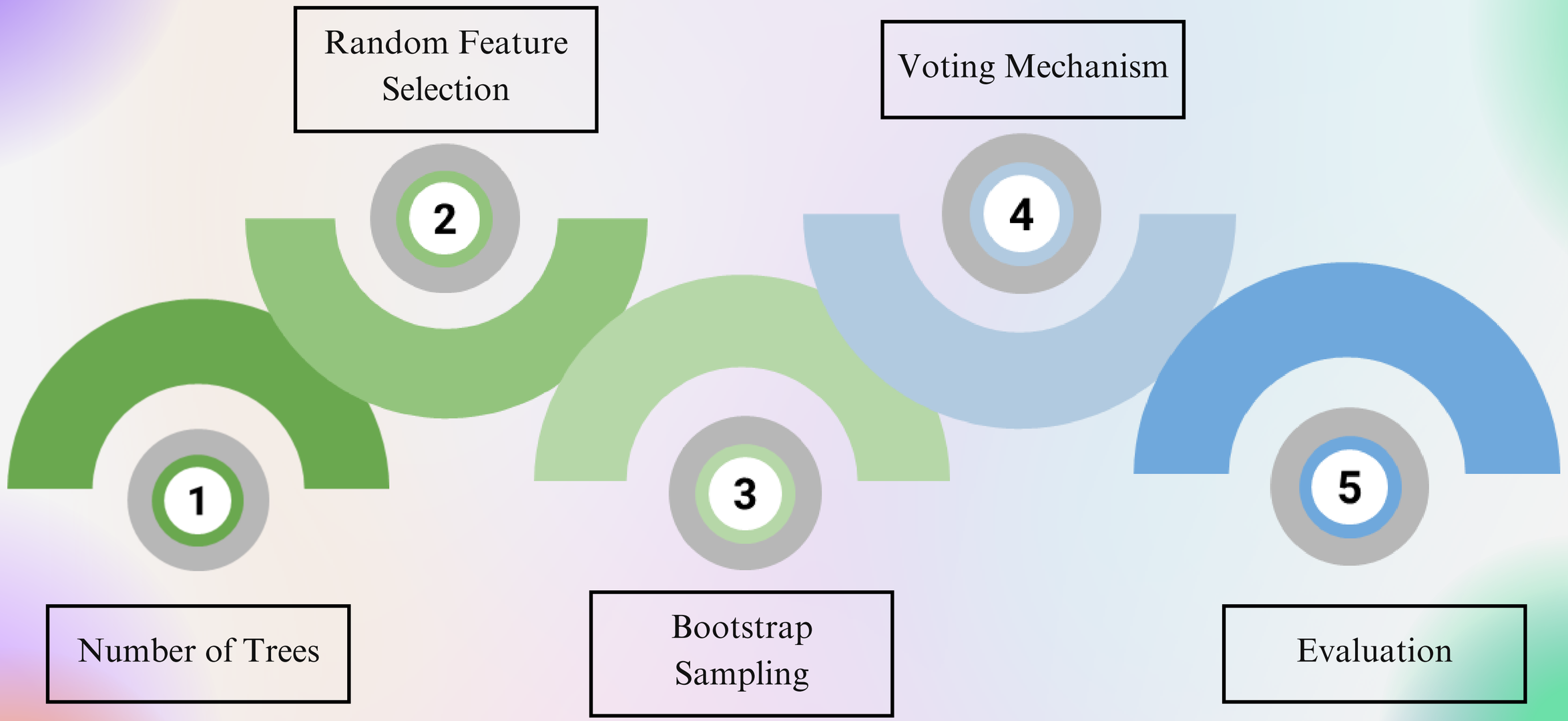
```
[26] # Evaluate the model
     loss, accuracy = model.evaluate(X_test, y_test_nn)
     print(f'Test Accuracy: {accuracy * 100:.2f}%')

     3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8954 - loss: 0.6428
     Test Accuracy: 88.46%
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.89 | 0.85 | 0.87 | 20 |
| 1 | 1.00 | 1.00 | 1.00 | 19 |
| 2 | 0.86 | 0.95 | 0.90 | 19 |
| 3 | 0.79 | 0.75 | 0.77 | 20 |
| accuracy | | | 0.88 | 78 |
| macro avg | 0.89 | 0.89 | 0.89 | 78 |
| weighted avg | 0.88 | 0.88 | 0.88 | 78 |



# Random Forest Steps:



# Building the Model & Model Evaluation

| Training: 80% | Testing: 20% |
|---|---|

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42, stratify=y)
```

*Training the model*

```
[ ] # Initialize and train the Random Forest model
    rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
    rf_model.fit(X_train, y_train)
```

*Make Predictions*

```
[ ] # Make predictions on the test set
    y_pred = rf_model.predict(X_test)
```

*Model Evaluation*

```
[ ] # Evaluate the model
    print("Accuracy Score:", accuracy_score(y_test, y_pred))
```

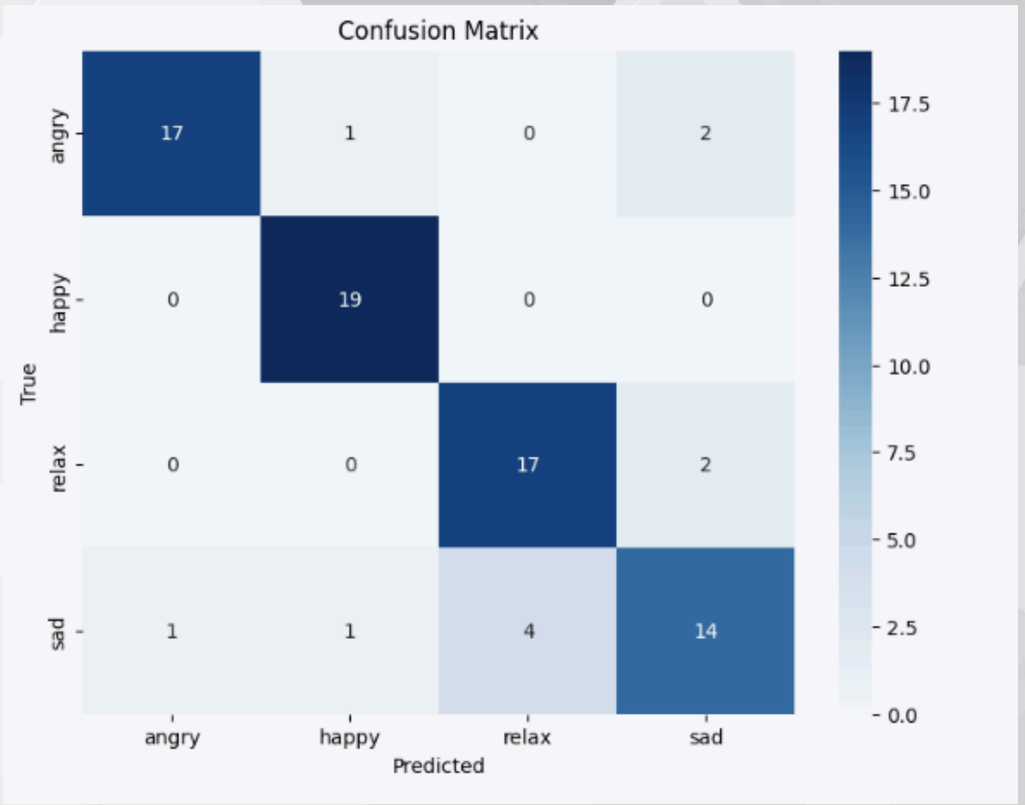
```
Accuracy Score: 0.8589743589743589
```

# Confusion Matrix & Classification Report

```
Classification Report:
              precision    recall  f1-score   support

       angry       0.94      0.85      0.89        20
       happy       0.90      1.00      0.95        19
       relax       0.81      0.89      0.85        19
         sad       0.78      0.70      0.74        20

    accuracy                           0.86        78
   macro avg       0.86      0.86      0.86        78
weighted avg       0.86      0.86      0.86        78
```



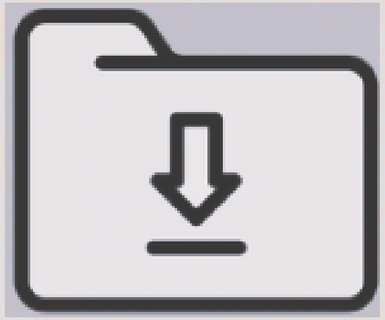
# Flask Web Application

Upload

Extraction


Result

Saving



ML Model

# Feature Extraction

Key Extracted Features:

RMS Energy
MFCC
Spectral Centroid
Tempo
Zero-Crossing Rate
etc.

These features represent:

Energy
Rhythm
Timbre
Tempo
Zero-Crossing Rate
etc.

# Prediction Process